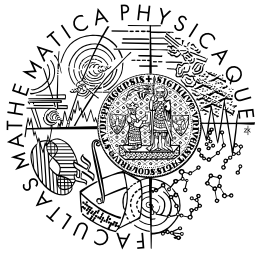

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Jan Pavlovský

Effective visualization for interactive video exploration

Department of Software Engineering

Supervisor of the bachelor thesis:	RNDr. Jakub Lokoč, Ph.D.
Study programme:	Computer Science
Study branch:	General Computer Science

Prague 2017

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ..... signature of the author

Title: Effective visualization for interactive video exploration

Author: Jan Pavlovský

Department: Department of Software Engineering

Supervisor: RNDr. Jakub Lokoč, Ph.D., Department of Software Engineering

Abstract: In this thesis we introduce an innovative approach to visualisation and search results presentation for large video collection search and browsing. The general problem of video search is analysed and discussed in comparison with other current software tools and methods used for video search. A specific visualisation method and algorithm for its generation is then proposed and discussed. We evaluated the methods both, empirically and by a user study. Based on the results, we chose the best possible algorithm settings for interactive video search and applied them. A simple experimental software tool implementing the proposed methods is developed focusing on the visualisation components.

Keywords: video search, exploration, visualization, multimedia browsing, deep learning

I would like to express my dearest gratitude to anyone and everyone who helped me in creating this thesis.

Namely I thank my supervisor Jakub Lokoč, Ph.D. for great guidance. I would never get to this point without him. I also thank Gregor Kovalczik, who is coworking on the project, for designing the semantic feature extractor and extracting the data from TRECVid dataset.

Last but not least I thank NIST and organisers of the TRECVid and VBS competitions for the datasets used in this work.

Contents

1. Introduction

The amount of video content created every day has been rapidly growing for the past decades and nothing suggests that it would change in near future. And at the same pace grows the need to analyse the wildly expanding video collections and search in them.

Video search has incredibly wide application possibilities, ranging from cinematography industry and video editing all the way to law enforcement agencies in need to find clues in the public cameras footage. And one other very critical use that has been gaining a lot of popularity recently is in medical applications implemented in modern examination methods.

There are multiple scenarios to encounter in all of these use cases. People need to search specific scenes in the video collections, they need to decide whether the collection contains any video of some arbitrary object or just generally explore the contents. In response to that, many challenging problems arise.

When searching through a video collection, there are many tasks one can face, such as scene detection, duplicate detection or specific object (such as person) or event search. Our primary interest is in so-called Known-item-search (KIS). In KIS, a user is looking for a specific scene he remembers, either visually, or by some semantic information (e.g. action happening, environment...). Besides specific KIS, one can search for a more general concept, for example, any "conversation on the beach during sunset". In this scenario, the goal can be to either find at least one scene with the described content, or all the scenes matching the description.

In all of these cases, the user usually does not have any direct way to query the video database[1] for the scene he is looking for. And that is the situation in which interactive exploration and especially effective results presentation come in place. Human operator aided search is still necessary for correct recognition of many abstract concepts, like sequences of events, or specific shots among many similar ones. And even for the tasks where it is not entirely necessary, user-driven interactive search can be often more effective in finding the right target (primarily because it can be difficult to formulate more complex queries for automatic search engine).

For the tasks presented, both computational and operational complexity are still very high and bring up many open and difficult problems. The whole area of video analysis has already branched into many more specific fields, e.g. video retrieval, video exploration, shot detection and so on. An excellent summary of recent work in all various fields and their differentiation can be seen in [1].

[1]The terms video collection and video database are both used in this work and are essentially interchangeable. However, collection is primarily used to refer to the actual content, while database is used in context of the video analysis methods to refer to the video data in computer memory that are being used for search purposes.

1.1 Structure of the thesis

In the first section, we are going to acquaint the reader with some of the essential theoretical basics used for video retrieval, especially the distances used for similarity measurement. We will also introduce methods used for storing video data and work with them. We will focus mostly on topics relevant to the contents of this work.

After that, we will briefly discuss other related work that has been conducted in the field recently.

We will then present the video search field as a whole and discuss its most important concepts. We will make some decisions for our system, reason a bit more in depth why it makes sense to study the interactive search methods. And we will shortly describe the retrieval model used in our system.

Next, we will present the more specific problem of video search this thesis focuses on, that is interactive video exploration and search results visualisation. We will perform a series of automated experiments and conduct a user study with the goal of supporting our hypotheses and methods. We will then compare results gathered from both.

In the last section, we will present the software tool implementing discussed methods. We will get a bit into the details of its implementation.

2. Preliminaries

We feel that it could come in handy for a reader that is not so well acquainted with video analysis to begin with a brief explanation of algorithms and mathematical methods we will be using as basis of our work. We will begin with a brief explanation of how video data are usually stored and how can we represent them in-memory for browsing purposes. We will present the ways to represent videos and their individual frames in effective ways for the search purposes. Then we will introduce how similarity among them can be measured and what means are most often used.

2.1 Video representation

Videos and especially their large collections commonly reach very extensive data sizes and it would be difficult to simply store them in computers. So encoding and compression is usually applied to store the video data on our computer's hard drives. There are many video codecs[1] currently in use. The most common standard for codecs is MPEG-4 and its ".mp4" format extension. MPEG is a working group of ISO and specifies the general parameters all codecs implementing MPEG should keep up to.

MPEG standard codec stores the raw images as a sequence of I, P or B frames, where I-frame stores the whole compressed image, while the latter two employ differences to reference frames. In our work, we consider so-called key-frames[2] representing selected and already decompressed frames (images).

More interesting for us is the representation of decoded video data in-memory as those are the data we will use for browsing. The first option is to load the data into memory in the same way as they are stored on hard drive. However that is probably not very practical. Although it saves space, it would waste a lot of computational time calculating the individual frames of video when they are needed.

Another option is to store the raw image data of certain video frames, or key-frames. We could stick to extracting the key-frames used by video codecs for storage (so-called I-frames in MPEG), but then we would have very little control over which key-frames are actually extracted.

So the approach we are gonna take is to extract key-frames at a fixed time interval from each other. This task may become quite computationally difficult. However, it has the inevitable advantage that we have raw image data in memory and we have exact control over which key-frames we will use. Also using this approach, we can partially correct motion blur during the extraction time and have clearer images saved afterwards.

The open-source software FFmpeg is used for extraction of the bitmap images

[1] CoDec is a commonly used abbreviation for COder-DECoder software for video that is able to decode compressed video data as well as encode raw data into a compressed video file.

[2] key-frame: Raw image data exported from a video frame at a specific time

from the video. It is distributed under the LGPL licence. [2] The FFmpeg dynamically linked library for .NET is available, but is not very stable as we experienced during its usage. So the choice we stick to is to call the batch processing FFmpeg.exe program as a sub-process in our application and export the bitmap data into a file.

Another problem encountered when saving videos is how to store audio data. Modern codecs implementing MPEG, such as .H264 provide this as an all-in-one solution and store the audio data together with visual data in so-called interleaved model. We will not be using audio data for purposes of this work.

2.2 Similarity measures

When querying the video database, we will stick to analysing the key-frames by their real visual content rather than any other information (such as names or annotations). Using this approach, we are generally speaking about content based image retrieval(CBIR).

For interactive search and especially for CBIR tasks, we need some tools to measure the similarity of different video key-frames to each other. For that, we need to store some kind of meta-data with each key-frame we use. The reason for that is there are no conveniently efficient methods currently known to compare similarity directly between 2 bitmap images.

We could simply compare each pair of corresponding pixels (applying some distance function to the colour values of the pixels, as points in R^n). However that is very error prone, because a slight motion of the objects onscreen could make a great difference in the colour of the specific pixels. And it is also computationally very expensive and in terms of time complexity unusable for our algorithms. Above that, it would also bring another problem of how to compare images of different dimensions.

The computational price could be lowered by comparing only a fixed number of specific pixels, however, it does not help with the other problems and also brings more difficulties, like which pixels to choose.

2.2.1 Distances and metrics

The dissimilarity metric to compare 2 images is of utmost importance for effective content-based retrieval. The metric generally gives us the distances between pairs of vectors in the same vector space. When we represent the images with some high-dimensional vectors (see below), we can set the "dissimilarity" of two images as the distance between the vectors representing them in the given vector space.

There will be 3 different vector spaces representing images used in this work, each of them built around different dissimilarity[3] measures. And in each of these

[3]A reader can see both terms, "similarity" and "dissimilarity" throughout this work. Though the term "similarity measure" is generally more common among the CV community, all of the distance function used actually measure dissimilarity, they return 0 for the same images. And

spaces, we have more option over which distance function to choose as the used metric. There is usually a direct proportion between the precision and complexity of each metric, so we will have to find some kind of compromise between them.

2.2.2 Colour difference

One of the simplest methods to measure the similarity of 2 bitmap images is, as mentioned in the previous paragraph, comparing the Euclidean distance between RGB colour of pixels in both images. Because of time complexity (proportional to the count of pixels, commonly reaching numbers as $10^6 - 10^7$ nowadays), we simply cannot calculate the distances for all pixel pairs. And we do not want to solve the decision problem of choosing which to compare, that is a topic for another work.

So we can simply downsize the images. There are many methods for high-quality image resizing out there, but for our purpose will be sufficient the simplest one of calculating the arithmetic average of colours of neighbouring pixels.

2.2.3 Colour Signatures

Another way to represent an image is a colour feature signature. A signature is, briefly said, a set of so-called "centroids" (essentially a colourful circle with given position, colour and radius), each representing a distinct colour feature of an image, or single frame from video. Each of these centroids contains 6 numeric values: the x,y coordinates in the image space, 3 colour values representing the colour space (could be in RGB, Lab or HSV depending on the implementation) and the weight representing the significance of the centroid.

That would suggest that we would create 6 times number of centroids" dimensional space for comparing images. Provided that the algorithm for extracting the signatures from bitmap images is sufficiently good, the signatures provide a good simplification of the visual data contained in the image. A good signature extraction algorithm must fulfill two primary criteria, it must represent the visual properties of the image well and it must decrease the data necessary to store it as much as possible.

Definition. Feature Signature of an image

Given a feature space F, the *feature signature* S^o of a multimedia object o is defined as a finite set of tuples $\{\langle r_i^o, w_i^o \rangle\}_{i=1}^n$ from $F \times \mathbb{R}^+$, consisting of representatives $r_i^o \in F$ and weights $w_i^o \in \mathbb{R}^+$

The problem arises when you want to compare the signatures to each other.

Firstly, there is not any fixed number of centroids to be in a specific signature. Defining such size would weaken the expressive strength of the signatures, which is not desirable. And secondly, the graver problem with comparing the signatures is that there is also no given order in which the signatures are given. Therefore to

the higher the value returned, the more dissimilar the images are - so in effect the distance is directly proportional to the dissimilarity of images.

get a reliable metric for the signatures, we would have to compare each centroid from one signature to everyone in the second one. More on the distance function we use for comparing signatures will be covered in the following section.

For extracting the colour signatures from bitmap images, we use the KNN clustering algorithm as proposed in [3] and [4].

2.2.4 DCNN feature vectors

The last but not least tool used to measure the similarity between individual keyframes covered are the Deep Convolutional Neural Networks (DCNN) and feature vectors extracted by them. A neural network is, generally said, a modern machine learning method with very good results in extracting semantic information from visual data. More specifically DCNNs are a special type of neural network that are inspired by the animal's visual cortex (the part of brain responsible for visual perception), and are created specifically for the image recognition task.

For the purpose of this work, it is enough to understand that DCNNs are a machine learning method for extraction of semantic information from images and the are able to extract a feature vector $\mathtt{v} \in R^n$ from an arbitrary image.

DCNNs should capture what the image really contains and what is happening on it, in a way similar to how human brain perceives it. Their most significant property from our point of view is that the individual values of a vector do not have a special fixed meaning. The network learns how to recognise objects during the so-called training phase before the deployment for use.

We use VGG network proposed in [5] and compile it using the Caffe framework [6]. The feature vectors are extracted from the 7th (penultimate) layer of the network and their dimension is 4096.

If the reader is not deeply interested in learning how the DCNNs work, for purposes of this work will be sufficient to understand that we use the DCNN to extract the feature vectors as descriptors of bitmap images. These feature vectors consist of 4096 floating point values. By measuring the distance (L2, cosine...) between these vectors in 4096-dimensional vector space they generate, we effectively measure the dissimilarity of the images the features were extracted from. In theory, the closer the features are, the more semantically similar the images should be (the more similar information is contained in them). The modern networks, such as VGG implemented in our tool, have a sufficiently good correlation with this perfect scenario.

2.3 Distance functions

The feature vector spaces presented , when combined with a distance function, generate an n-dimensional metric space, where n is the size of a feature vector they use and could be varied throughout different implementations. A little problem arises with the signatures - they do not form a metric per say (do not fulfill the formal definition of a metric), buy it should be no harm to speak about them in

the same way as about the other metrics, while keeping in mind this fact.

A point in the feature vector space, therefore, corresponds to a single image. So we can measure their distance. Following is presented a brief overview of distance functions in use (those we have implemented in our system):

Definition. L2 distance or Euclidean distance

Given 2 vectors: $\mathtt{u},\ \mathtt{v} \in R^n$, the L2 distance $dist_{L2}$ is given as

$$dist_{L2}(\mathbf{u},\mathbf{v}) = \sqrt{(u_1-v_1)^2+(u_2-v_2)^2+\cdots+(u_n-v_n)^2} = \sqrt{\sum_{i=1}^{n}(\mathbf{u}_i-\mathbf{v}_i)^2}$$

Definition. Cosine Distance

Given 2 vectors: $\mathtt{u},\ \mathtt{v} \in R^n$, and angle θ between them, the Cosine distance $dist_{cos}$ is given as

$$dist_{cos}(\mathbf{u},\mathbf{v}) = 1-\cos(\theta) = 1-\frac{\mathbf{u}\cdot\mathbf{v}}{\|\mathbf{u}\|_2\|\mathbf{v}\|_2} = 1-\frac{\sum_{i=1}^{n}\mathbf{u}_i\mathbf{v}_i}{\sqrt{\sum_{i=1}^{n}\mathbf{u}_i^2}\sqrt{\sum_{i=1}^{n}\mathbf{v}_i^2}}$$

Definition. Manhattan Distance

Given 2 vectors: $\mathtt{u},\ \mathtt{v} \in R^n$, the Manhattan distance $dist_M$ is given as

$$dist_M(\mathbf{u},\mathbf{v}) = \sum_{i=1}^{n}|\mathbf{u}_i-\mathbf{v}_i|$$

2.3.1 Perceptually Modified Haufdorff Distance

As mentioned in the previous section, comparing signatures of images is significantly more complicated as they do not have a fixed size. We use the Perceptually Modified Hausdorff Distance as proposed in [7] for this task. It is implemented by computing both directional Hausdorff distances and using the maximum of these two as the PMHD metric.

Definition. Perceptually Modified Hausdorff Distance

Given two feature signatures $S^o = \{\langle r_i^o, w_i^o\rangle\}_{i=1}^n$ and $S^p = \{\langle r_i^p, w_i^p\rangle\}_{i=1}^m$ and a distance function δ, the Perceptually Modified Hausdorff distance is defined as:

$$PMHD(S^o,S^p) = \max\{h(S^o,S^p), h(S^p,S^o)\}, where$$

$$h(S^o,S^p) = \frac{\sum_{i=1}^{|S^o|}[w_i^o \times \min_{j=1}^{|S^p|}\frac{\delta(r_i^o,r_j^p)}{\min(w_i^o,w_j^p)}]}{\sum_{i=1}^{|S^o|}w_i^o}$$

2.4 Metric usage options

Various combinations of vector spaces with different distance functions can be made to create metrics achieving the best results in different situations. For the biggest part, the L2 metric is used for the Color similarity, and Cosine distance for comparing the DCNN feature vectors. That is primarily because the L2 metric would be too slow for calculating the distances between 4096-dimensional feature vectors.

The cosine distance has been chosen as the most effective one because of its implementation optimisation possibility. The norms of all the feature vectors can be pre-computed during the loading phase and stored inside of the key-frame objects. This way, only one multiplication has to be done on the divisor side, instead of 8192 (2 feature vector sizes $\in R^{4096}$) second power calculations and additions.

3. Related Work

There is a lot of interesting tools for video browsing and CBIR in videos around. However, vast majority of them are currently in the phase of research projects, there is little commercial software. That is very motivating for the development of these tools, as there is a lot of space for improvements and video analysis generally seems to be still gaining more and more popularity.

Many research teams focusing on video browsing compete in the VBS(see below) competition, just as the team from Department of Software engineering of MFF UK. Here we provide a brief overview of the interesting tools presented in the last year, VBS 2017 organised as part of 23rd International MultiMedia Modelling conference in Reykjavik, Iceland.

An inspiring work focused on interactive search possibilities as well as on the collaboration of a team of users in completing a search task can be found in [8]. The team implemented a large statically generated image map of all the scenes of video collection and let the users to interactively browse through it. The improvement was that the system was run on a server and all the users connected to the system could see what part of the collection are the others browsing. The search engine was also optimised not to display results someone else is already browsing. This software also focused on the interactive switching of different visualisation forms.

Also a very interesting work about video search visualisation methods published last year is [9]. In this storyboard-based system, the algorithm first detected the scene cuts and then clustered the individual scene representatives based on color distribution. The result is a visualisation allowing to both, display time context of the scenes and sort them visually.

The core of the retrieval model implemented in the software implemented as part of our work is inspired by the Video Hunter software, lastly published in [10]. We implement the same neural network model for semantic features extraction, as well as the same signature extractor.

Besides VBS 2017, we gained a lot of inspiration from [11] published in 2015 by Nico Hezel Et al. They implemented a system able to sort large collection of videos into an image map of representatives for the scenes. They were trying to sort the graph so that is would be easy to visually orient in the visualisations. The whole graph of scenes was created once for the whole collection and then during the browsing, user was allowed to traverse the graph in order to search the collection.

3.1 Competitions

There are many competitions organised to benchmark and compare the results in video search tasks. Among the most established ones are the TRECVid organised by NIST (National Institute of Standards and Technology), Video Browser Showdown (VBS) organised as part of International MultiMedia Modelling conferences

(currently collaborating with TRECVid) and Mediaeval.

Their primary objective is to push the state-of-the-art video browsing tools and help with the interaction between teams working to improve them. However being competitions, they also benchmark the performance of current software tools developed by individual teams. The two factors evaluated are the time it takes to find the goal and the precision, whether the correct target scene is selected.

The tasks participating teams are faced with at VBS are primarily KIS, more specifically visual KIS and textual KIS tasks. In visual KIS, the operator is presented with a short scene from an arbitrary video from the collection. His task is to memorize the key features of the scene and find it using his tool. In textual KIS, the target scene is only described textually and the user is to find it anyway. At present time, these tasks generally present greater difficulty due to the semantic gap between text and image, that is still huge for computers.

A team from Department of Software Engineering of MFF UK has been competing in VBS since 2014 and was able to win twice and reach the 3rd place in 2017 with software initially developed by Mgr. Adam Blažek under the supervision of Jakub Lokoč, Ph.D.

4. Video Search

4.1 Video retrieval in general

Video retrieval tackles the problem of search for an arbitrary scene in the video collection or its part. When querying the video database, the first question that naturally arises is what to ask for, how to build the query. The most usual way for creating any search queries generally is to search by keywords. However, there are many problems with keyword search in video databases, such as choosing the right combination and specificity of words, and primarily the problem how to annotate the video.

To the problem of connecting the image data with its textual description is usually referred to as the semantic gap[1] and its difficulty is one of primary reasons interactive exploration methods like the ones we present are necessary. Machine learning methods come in handy in overcoming the semantic gap. Especially by use of the supervised learning[2], we can teach the computers (specifically the neural networks in our case) to create keyword annotations for the video collections automatically. With those generated, we would be able to search for scenes by keywords.

Many methods pursuing this task have been proposed in recent years, however none of them is getting any near to how humans would manually annotate the video so far (both qualitatively and quantitatively). Even for annotating single images(representing key-frames or simply static pictures), the annotations created fully automatically tend to be, especially quantitatively, much less expressive than the ones a human user is able to come up with (machine learning is not able to find so many abstract concepts a human operator car).

Another option we have is to use the so-called Query by example (QBE), to present the database with an example image, be it a picture we found on the internet, other video key-frame or just a sketch we painted straight from our memory. The system then usually finds the "k nearest neighbours" (KNN) of given example in our collection by given similarity measure[3]. The variable "k" here refers to the number of results we want to obtain from the database query. It usually corresponds to the number of key-frames the visualisation method in use is able to display, and therefore will be use-case dependent.

The work-flow described above allows us to browse the collection interactively and to gradually filter and refine the retrieved results by performing multiple QBEs in sequence. Using this method, we are generally speaking about CBIR.

[1]The semantic gap represent the difference between two different representations or understandings of the same object. Typical examples are the difference between 2 languages or, in our case, between an image and its textual description.

[2]supervised learning is a way of machine learning, in which the computer is supplied with a set of input and corresponding correct outputs for all of them and seeks the function returning correct outputs for all the inputs

[3]Individual frames or scenes of the video are usually represented by a vector in some metric space, so that we can measure the similarity between them

The results presentation is of great importance for CBIR because it allows the user to quickly see whether the results displayed contain his target and lets him intuitively select the examples for the following query.

There are many tools and approaches for searching video collections, and a lot of effort is being put into improving these throughout the computer vision (CV) community. However, browsing very large video databases is still an extraordinarily challenging task both, for fully automatic systems and for humans. There are some current software tools out being developed nowadays. However we suppose they focus more on results retrieval than specifically on the results presentation. This work will be concerned primarily with the presentation of the results generated by arbitrary retrieval system (although a simple retrieval model was implemented), more specifically images representing individual key-frames from the video collection.

4.2 Motivation of this work

As presented in the previous paragraph, a user-centered search is a very powerful tool in content search inside large video collections. It seems to be very difficult to fully replace it by automatic search engines in overcoming the semantic gap.

The main purpose of interactive search is to effectively solve the KIS tasks when the user is looking for a specific scene but is not able to perfectly describe it. Or he just remembers that the collection contains the scene he is looking for, but can not remember it exactly unless seeing it. In either of these cases, displaying many relevant results for the queries and letting the user decide which of them are similar to what he is looking for is essential for finding the target scene quickly and efficiently.

The necessity of some example image or scene to present the system with for CBIR significantly limits its possibilities of usage. In many scenarios, the user is able to recognise the target scene or a similar one when it is presented to him, but has no example and no other way of generating an example image to query for. And that is one of the scenarios we focus on. We aim to solve this problem by allowing the user to begin the search by displaying many random video frames and choose the example for the next query out of them, or generate another random set if no suitable one is found.

We deduced that the more results we present to the user in response to his query, the quicker he would be able to find what he is looking for. However, that holds only under the condition that the user is able to keep track of the displayed results and quickly recognise the visual data presented. If too many images were to be displayed, they could be simply rendered too small to recognise on the device user is using. And even if that was not the case, the user could have a problem with processing so many visual precepts at once.

It has been shown in [12] that human brain can only apprehend so many precepts at once. It is estimated that for bitmap images, the limit for an average person is 20-50 images at once [13]. In our domain, it means that if we would try to show the user more images than this upper limit, he would not be able to

notice all of them immediately. That would impair the overall ability to recognise the result and significantly slow down the whole search process.

If, however, the displayed images were to be well-organised and arranged, we assume that the user will be able to find his way around the display much faster and in fact perceive more images at once. Results of [14] support this hypothesis. The proper arrangement of displays created allows for faster exploration which will positively impact browsing speed and quality, especially in very large video collections. If we would be able to reach several hundreds of frames at once, instead of the 20-50 images per display limit, it would improve the experience and effectivity distinctly.

Our original inspiration was ImageMap system introduced in [15]. The system was able to sort millions of images coming from various sources into the grid by their visual similarity. It also allowed the user to interactively browse the results with different Level of Detail (LOD) settings. Its extension for video data was introduced in [13]. In this work, we present a follow-up of that system.

The major flaw we see in Graph-based video browser is that it calculates the image grid statically for the whole dataset and then only chooses relevant parts (and adequate LOD) to show the user in response to a query. We aim to improve this by calculating the result displays dynamically on the run for every query. If we sort image results for every query by their visual and semantic similarities, the resulting displays should be much easier to grasp for an average user because they will contain key-frames more closely related to his query. Following this approach, we believe the user will be able to efficiently and quickly browse the database.

4.3 Interactive search

The primary goal here is to make the general purpose video search as efficient as possible. We presume that the user is looking for a scene he remembers or has some information about and is able to recognise it when it is presented to him. The user usually wants to query the database with some initial query method (be it text or example image) and then he can interactively browse the collection.

When there is no way to create the initial query, displaying many random shots from the collection is another option. That is the only option for initializing the search in the software attached to this work, as its primary purpose is to test out and study the interactive search and visualisation methods, not the search engine itself.

We aim to minimise the time and effort user needs to find his target scene. In that sense, we need to minimise four factors with most striking effect on the search:

- The video database query time.
- The display generation time.
- The time user needs to orient in the display and find relevant results in it.

- Number of steps (DB queries) user needs to find the target scene.

In this work, we will be focusing primarily on the latter 3. The effectivity of the actual search in the collection and query results retrieval is naturally very important. However, there have been proposed many methods for that in related fields. We will consider the search results as given, and also in our current implementation will employ only very simple search engine. In the production quality software, proper indexing and search acceleration would be necessary.

The methods proposed in this work are primarily focusing on the KIS task. However, they are mostly applicable for other kinds of search tasks as well. Following are the 3 factors we would like to improve described in more detail.

4.3.1 Efficient display creation

Generating a good quality visualisation so that the user will be able to quickly get around it and find relevant data is of utmost importance. However, the efficiency of this task if of at least the same importance. When exploring a very large collection of video data, it is not possible to let the user wait. Besides of the actual efficiency, user friendliness can be seen as another reason for that, as perceptible labs are not expected in modern production quality multimedia exploration software. The queries need to be calculated almost instantly and results displayed in time less than half a second. In the best scenario, it should be even faster.

4.3.2 Display quality

On the other side of speed is the quality of generated displays. It is important to keep everything well ordered and arranged so that the user does not lose track of what he is seeing. Keeping similar frames together is the most important part here. And in order to really improve the efficiency of the whole search task, images need to be well sorted both, visually and semantically.

4.3.3 Search steps

The last thing that significantly affects the overall time it takes to accomplish the search task is the number of large steps necessary to take. By large steps, you can understand any action that takes a nontrivial amount of time to finish, changes the display completely or even starts the whole search process from scratch.

The step with the highest negative effect on the efficiency of the methods proposed here is the video database query, be it by example image, text or any other method. Evert single query slows down the search process significantly. The query takes some time to compute and display, afterwards the user has to get around a completely new display which also takes a nontrivial amount time to generate, even using very effective display methods.

There are multiple paths we can take to lower the overall number of steps. The simplest and very natural way would be to display as relevant results for every query as possible, where the user could find his target in the first few results displayed. That can be accomplished by well-designed and effective querying. However the search engines are not perfect and so this is usually not possible.

This thesis focuses mostly on another path possible, displaying as many results as possible for every single query and enhancing the interactive browsing among them.

5. Video Content Visualisation

When interactively browsing a large video collection, an utmost important component of the system used is the data presentation. As already stated in previous chapter, it determines how fast the user will be able to get around the data presented and in effect browse the collection. In this chapter, we discuss different methods that are used for video search results visualisation. After that, we will present an innovative way to display the data using dynamic large key-frame grids.

The simplest and most natural way to display a video is chronologically unfolding the video frames according to the time context, one frame after another. However as the collection gets bigger, this option quickly gets very slow to use because you can see so little of the whole collection. This becomes even greater problem especially for high count of short shots (which is a very common scenario when filming a movie).

So other methods able to display more content at once come into light. There has been quite a boom in interactive video search since VBS competitions have shown its actual potential. Human-aided interactive search can usually achieve much higher precision in KIS tasks. By precision, we mean the mean probability of choosing the right target scene in a search task (especially when the user is not entirely sure what he's looking for or it is not even absolutely specifically given - general target search).

While there have been some methods targeting this problem proposed already, we feel like none of them has really overcome the problem. There are few key paths we can take to expand the number of images we are able to display at once. Among those are:

- Displaying multiple scenes in response to a single query.
- Displaying results from multiple videos.
- Compacting the video frames rendered so that we can display more of them - longer time span.
- Displaying a large number of individual frames from various videos and scenes.

Figure 5.1: Example of video data expanded into stripe of key-frames expressing the time context of the scene

Each of these approaches is useful for different scenarios and even during one search task, different methods can be practical to use during various phases of

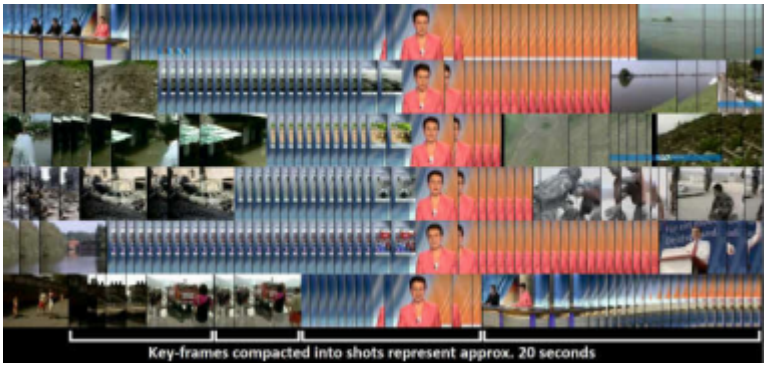

Figure 5.2: Example of compacted video frames with multiple results displayed. View adopted from [4]

the browsing. Assuming we are able to effectively switch between more visualisation methods, interaction between more of them can really enhance the search effectivity.

Generally speaking, the less the user knows about what he is looking for, the more results shall be displayed to give him a better overview of the whole collection contents. And later when the user has the idea of the collection contents and better understanding of his target, he can switch into displaying lesser results in more detail.

So we also need to solve the problem of how to quickly and effectively interact between multiple visualisation methods. For example, when we want to switch from a large display showing many frames from different videos into the single scene time context view, we need to display the scene containing the key-frame we were focused on and possibly also its neighbours (in multiple rows/columns). However, as individual key-frames can be from distant parts of the collection, we need to choose the ones to display carefully not to overwhelm the user with a set of totally different scenes.

5.1 Presenting large displays

As the size of the video collection targeted gets bigger, the chance that the retrieval model can hit the target scene in first few results decreases inversely proportionally. That is true especially when the user has only vague idea of what he is looking for (for example in textual KIS).

To increase that chance, we can display more results to show wider array of scenes. That way the probability of displaying the correct result will get higher assuming the same retrieval model. However, the detail of the images displayed deteriorates with their increasing count. And when we display too many results, the user could get lost and would not be able to recognise the presented key-frames at all.

In our method, we would like to find the sweet spot just under that level, displaying as many key-frames as possible at first, so that the user will still be able to keep track of them. And after that, the user would be able to browse

through many different displays interactively and select example images for next QBE from the displayed results. In figures 5.3 and 5.4, we can see examples of such large displays.

Figure 5.3: High count of random video key-frames arranged in a 2D grid

Figure 5.4: Few well ordered video key-frames arranged in a 2D grid

In the displays generated, we want to keep the similar images together in clusters and the obviously different ones as far from each other as possible. So in other words, we aim to preserve the complex high-dimensional relationships between images in the low-dimensional space we embed them into. Most importantly, we would like to ensure that the key-frames keep the ordering of distances from all the others as similar as possible in both the spaces (image feature space and 2-dimensional visualisation space - the screen).

Also we would like these relationships between images to be well preserved both, visually and semantically. Imagine for example any arbitrary images of 2 apples, one green and one red, and a green pear. Can you tell which of those are the more similar ones? It would be the apples probably. However what if the images were very blurry for some reason. Blurry images can be caused for example the motion blur, very common when extracting single frames from videos. Then the two blurry green shapes would become much more similar to each other than to the third, red shape. That is why the pure visual information is also important, without any regard of what the object displayed really is.

There are many options for choosing the way we visualise the arranged results on-screen. As the space to embed into, we chose a 2D rectangular grid of images mainly because it should be the most approachable one for a human operator. Compared to others, for example 2D positioning without fixed grid, we presume that the individual key-frames are much better visible for the user in the grid. We assume that this is true primarily because they are well visually separated by the regular grid net.

For generating the displays described, we need to measure the overall similarity of all the key-frames to their neighbourhood. To measure this similarity locally is a trivial task, given the feature vectors for all key-frames and the distance function. We can simply sum up the similarity of an image with all its neighbours. And combining the visual and semantic similarity metric also is not very difficult task. In the simplest case, we can just add up both similarities and possibly also

normalise them to give a better aggregated idea.

However getting to the broader view, it is not so easy to define the similarity energy function globally. In the following section, we discuss few possible options we could use to measure the global energy(or cost function) of the displays.

5.1.1 What to optimise

We want to minimise the overall similarity sum of images next to each other in the grid. This suggests we are dealing with an optimisation problem. When dealing with optimisation, we need some sort of global energy function to minimise. And with our displays, it is not so obvious what the energy should be.

The first possibility is to add up local similarity sums of all the images in the grid with their neighbourhood. That would definitely give some idea of how close to each other are similar images locally at each place in the grid, however it would suffer greatly in expressing the global energy. For example, it is almost inevitable that more clusters with almost identical images can appear far away from each other using this cost function when increasing the size of the grid. The reason for that is that when the clusters are more distant from each other than the maximal distance the energy is locally compared for, their distance never gets measured.

The distance we are locally comparing to is a very important parameter of any algorithm emphasising the local similarity in any dimension. It is generally given by a distance from an object in a metric in use and generates a sphere around the object where all other objects contained are considered to be in its neighbourhood.

Summing up similarities of all the images in the grid is obviously out of the way, as this value would be constant in any arrangement of the grid with the given key-frames. However taking into account also the distance in the grid (e.g. Manhattan distance) and computing the difference between their normalised values, we get a nice energy function that expresses well what we want to achieve. For images far away in the original high-dimensional space, we want them to be far away in the graph. And for the ones near, we need them to come close to each other. More formally:

$$E = \sum_{i,j=0}^{n,m} |\delta(x_i, x_j) - L_M(x_i, x_j)|$$

- E stands for the global energy
- x_i are individual key-frames
- δ is distance function for any of the high-dimensional spaces representing the real relationships between images
- L_M is the normalised Manhattan distance in the 2D grid.

The problem of embedding high-dimensional data into low-dimensional space is actually a well-established field and there have been many methods proposed

targeting this problem. However, none of those is really well suited for this use-case. The reason for that is that most of the methods are able to embed the data well into general 2D or 3D spaces. However, it is very difficult to arrange the images into regular rectangular grid afterwards.

One of modern methods for high-dimensional data embedding is T-SNE, originally introduced in [16]. In T-sne, the global function to be minimized is defined as:

$$E = KL(P||Q) = \sum_{i,j=0}^{n,m} p_{ij} \log(\frac{p_{ij}}{q_{ij}})$$

That is the Kullback-Leibler divergence between two probability distributions which measures the divergence between 2 individual probability spaces. It represents surprisingly well what we are trying to approach, for all the objects in both spaces to keep the same ordering. While we used this measure for experimental purposes, as it nicely expresses our goal, it is not fit for practical use, as the calculation of probabilities from distances is very computationally expensive and essentially unnecessary for our methods.

What we did is to adopt calculating the logarithm of the high-dimensional and low-dimensional distance ratio, as it expresses one very important property. That it is much more important for the very similar images to be displayed near to each other as opposed to displaying the dissimilar ones far away.

The reason is that the worst that can happen when visualizing the images in the grid is that 2 clusters of very similar images get formed far away from each other. On the other hand, we do not mind that much sharper visual differences between individual clusters. The improvement fixing that is as follows:

$$E = \sum_{i,j=0}^{n,m} |\delta(x_i, x_j) - L_M(x_i, x_j)| \log \frac{\delta(x_i, x_j)}{L_M(x_i, x_j)}$$

This function nicely represents the global dissimilarity of the grid. However, for statistical purposes it has one very significant flaw, its absolute value is greatly affected by both grid size, and the absolute sum of high-dimensional distances between individual key-frames.

The function that we implemented in the end for measuring the global energy in the experiments is the Kruskal's stress measure, as stated in [17]. The cost is given as:

$$E = \sqrt{\frac{\sum (d_{ij} - \delta_{ij})^2}{\sum d_{ij}^2}}$$

The main advantage of this cost function is that, it is exceptionally stable across different dimensionalities, sample counts and the absolute distance sums because of its normalisation. Also, it is already accepted as a good measure for multidimensional scaling in a study regarding this problem [7].

5.1.2 Natural optimisation problem

That is for the formal definition of the grid stress function problem. However the purpose of our work is more of empirical nature, the aim is for the user to orient well in the displays. So a more natural way of viewing our problem is from the point of view of the actual human user. He wants to search quickly in our images grid. In order to do that, the grid should be organised in such a way that the user looks at it and is able to immediately recognise the important parts to him.

That being said, the problem of representing this idea in a machine readable way is a topic for a whole other work. From here on, we will assume the proposed Kruskall's stressfunction to be good enough to describe the situation and express how well organised the grid for the initial examination of the methods.

And for the empirical persuation of this hypothesis, as well as other methods used, a user study evaluating grids generated by different methods will be conducted. Afterwards, the result of the study will be compared with the experimental results with the goal of finding a correlation between them.

5.2 Generating the displays

The large image grids are used as main visualisation component for the software tool developed. So the most important part is how the grids are generated. We have proposed an algorithm based on the heuristic, that by gradually improving local similarities of many various areas in the grid, it is possible to minimise the global cost. The algorithm presented below is very similar to the ones in [13] and [15]. The primary difference is that we focus on the efficiency so that we are able to generate the image grids dynamically on the run. That also implies that the number of key-frames that need to be sorted is typically significantly lower than in the works mentioned above.

The core of the algorithm is iteratively selecting random pairs (or n-tuples) of the images in the grid. Then, the sum of similarity with their neighbourhood is calculated. The similarity is then calculated also with the neighbourhood of the other selected key-frame for both of them, giving all 4 sums (for both key-frames with both neighbourhoods)s. By summing the original sums together and the sums for switched key-frames together, we get a simple dissimilarity measure for both versions of the grid that we can compare.

$$NS_{kl} = \sum_{i,j=-n}^{n} \frac{\delta(x_{k-i}, x_{l-j})}{L_{max}(x_{k-i}, x_{l-j})}$$

- NS means the similarity of a key-frame with its neighbourhood
- x_i are individual key-frames
- δ is distance function for any of the high-dimensional spaces representing the real relationships between images

- n is the size of the neighbourhood[1] to compare the image to
- L_{max} is the maximal difference in any single coordinate

$$L_{max}(x_{ij}, x_{kl}) = max(|i - k|, |j - l|)$$

We calculate all of these distance sums and if the switched dissimilarity sum gives us a lower value than the original one, we switch the images in the grid. If not, we keep the grid the same. In either case, we continue with the next iteration by choosing another random pair until some terminal condition is met.

The terminal condition can be set in terms of different parameters. The simplest one is to specify the total number of iterations to take. Another option is to specify the maximal real time we let the algorithm to run. A more complex version could be to stop the algorithm when the improvements over some number of runs get too small to actually make any difference. Naturally, we can combine more of these approaches to get the best results possible in every situation. In the experimental section, we focus more on choosing the right option.

The main algorithm is formalized in the *Algorithm 1* figure.

[1]The neighbourhood can be of different shape than just a simple ball - a rectangle in 2D regular grid. However the one chosen to be used in our tool is the simple ball, so it is more suitable to formalize only this one, as it is also much easier to do.

Algorithm 1 Image grid generation

procedure GRID INITIALIZATION
  results ← query results
  grid ← initial arrangment of *results (2D array of key-frames)*
procedure DISTANCES CALCULATION
  $i \leftarrow 0$;
  $j \leftarrow 0$;
  for `i < length(results)` **do**
    for `j < length(results)` **do**
      distances[i, j] = DistanceFunction(*results*[i], *results*[j])
      $j \leftarrow j + 1$
    $i \leftarrow i + 1$
procedure MAIN LOOP
  while not(*terminal condition*) **do**
    $frameA \leftarrow$ random $frame$ from *grid*;
    $frameB \leftarrow$ random $frame$ from *grid*;

    $similarityNow \leftarrow \sum$`distances[frameA, neighbourhood(frameA)]` + $\sum$`distances[frameB, neighbourhood(frameB)]`;

    $similaritySwitched \leftarrow \sum$`distances[frameB, neighbourhood(frameA)]` + $\sum$`distances[frameB, neighbourhood(frameA)]`;

    if *similaritySwitched* $<$ *similarityNow* **then**

      $grid \leftarrow grid.Switched(frameA, frameB)$

5.3 Parameters and variants

The two parameters with the most striking impact on the performance of the algorithm are the total number of iteration it runs and the neighbourhood size (understand the number of key-frames in one's neighbourhood the similarity measure is calculated against).

A variation of the algorithm we also tested is to choose more images than only 2, compare all of them and then, we have 2 options how to proceed. Either we switch the 2 images that give us the highest improvement when switched. Or we permute all of the images evaluated into the permutation that gives us the best overall similarity. We benchmarked both these variants for choices of 3 and 4 random images.

Under the same time constraints (meaning for the display to be generated in under 500ms, more on this is covered in the experimental chapter), all these methods gave us worse results than the simpler variant with switching only pairs of images, so we did not include these methods in further experiments or the user study. The reason for that is that calculating the overall similarity of every image chosen with the neighbourhood of everyone is computationally too expensive and allowed us to run 2-8x fewer iterations depending on the settings (or the same number with equally lowered neighbourhood size). So the improvement that the global stress of the grid can get by rearranging more images than 2 at once does not pay off for the computational cost increase.

5.4 Key-frame selection and shot detection

With videos nowadays commonly reaching frame-rates of 60 fps (frames per second) and even much higher, another problem we face when generating video content visualisation is the selection of key-frames to be displayed. Even with the common frame rate of 30 fps, using every single frame would be simply too much both, for the computation, and for the user. He would be overwhelmed by the almost similar images of a shot taking place for example during just one second (often 30 almost identical frames).

The first level of key-frame selection is to export and analyse key-frames with specified time period between them. The rate of exporting 1 frame per second was chosen for the methods proposed. We assume from the empirical examination that 1 frame per second is enough information to detect most target scenes in standard video files, yet it provides us with a convenient way to reduce the amount of data worked with.

However, that is still too much data for scenes with longer duration. Even for 10 seconds long static scenes, the used dataset would contain 10 almost identical key-frames. So we need to introduce also shot detection to further decrease the count of key-frames used.

A simplified version of the algorithm proposed in [18] is used to detect the shot boundaries. Simplified because as we begin with extracting the key-frames only once every second, there is no enough information for a proper shot detection. It

would be more precise to define the approach as detecting sharp visual differences between every 2 and 3 key-frames in a row.

The distance is calculated for every 2 extracted key-frames in a row using multiple distance functions. The first measure if simply the sum of dissimilarities of the 2 key-frames. If the dissimilarity sum is higher that the threshold set, they are separated into different shots.

The second measure implemented incorporates the ratio of similarity between the chronologically previous key-frame and the following key-frame. A very high ratio means that one of the key-frames is very similar to the evaluated one while the other one is dissimilar. That suggests that the similar one belongs into the same shot, while the other one belongs to a different shot. So a cut is detected in the place with the higher dissimilarity.

6. Experiments

In the experimental part, we made variety of tests and experiments to determine which of the approaches are the best to take and to fine-tune the parameters of the main algorithm. As the purpose of this work is to make the video search as effective as possible and to lower the overall time it takes the user to find the target scene he is looking for, the primary constraint necessary to abide is the computation time. We set a kind of constraint, that the whole visualization for production use should be generated in under 500ms (half a second) considering usage on average personal computer as of 2017.

The reason for that is that if the time to get the results and generate the visualisation would be higher, the lag between the user initiating the search and the visualisation being displayed would become significant.

Because the total time will be used as measure besides the iteration count and other logical units, it is necessary to state the computer configuration that was used for the tests. All the tests were performed on an average personal computer around 5 years old, so result on modern machines should be significantly better.

The test configuration is:

- motherboard: ASUS P7P55d Pro
- CPU: Intel core i7-860 at 4x2.8GHz
- RAM: 16GB of DDR3 Kingston HyperX blu at 1600Mhz
- hard drive: Samsung 840 Pro series SATA SSD
- GPU: ATI Radeon HD 5850, 1GB GDDR5

The dataset the experiments were conducted on consists of the total of 800 key-frame grids generated by our application. Among those, all the significant algorithm parameters are varied to find the best possible settings and to show the differences between then. For the top level difference, of the 800 grids generated:

- 276 were generated using the semantic similarity (feature vector cosine distance)
- 258 were generated using simple color L2 distance
- 266 were generated using signature PMHD function

As the global similarity measure was used the Kruskal's stress function, as introduced in the Visualisation chapter, "what to optimise" section.

6.1 Results

The first important parameter tested is the neighbourhood size to be compared. The results presented further show that increasing the neighbourhood size improves the results quality distinctly, visibly the most out of all the parameters involved. However, the computational time necessary also increases exponentially with expanding neighbourhood. So setting the neighbourhood to the highest possible value (which is half the size of the grid - highest value that makes sense to use) is obviously out of the way, some compromise has to be found.

In figures 6.1, 6.2 and 6.3 you can see bar charts representing the average normalised overall grid stress value after the algorithm run. The iteration count was varied between 10 000 and 100 000 iteration to get the average value among different settings.

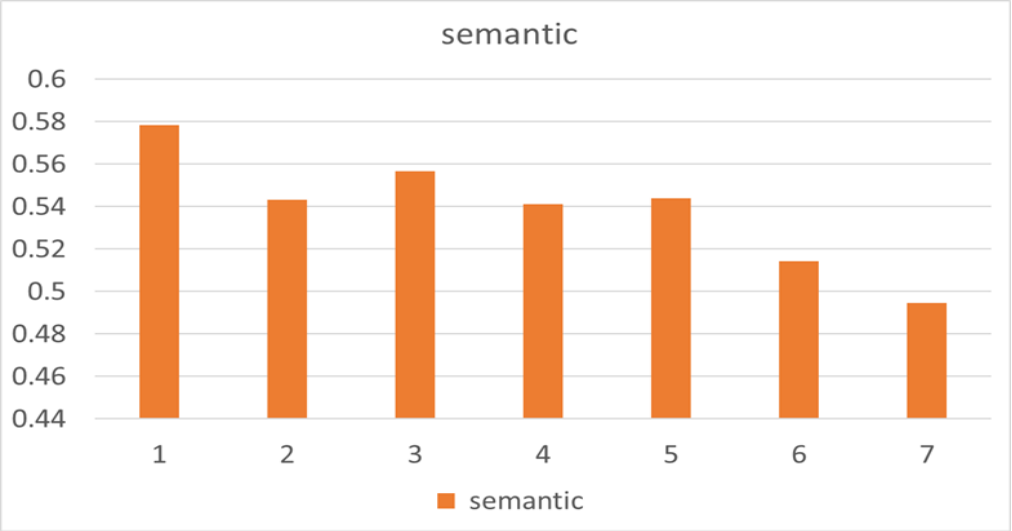

Figure 6.1: Neighbourhood size for semantic similarity

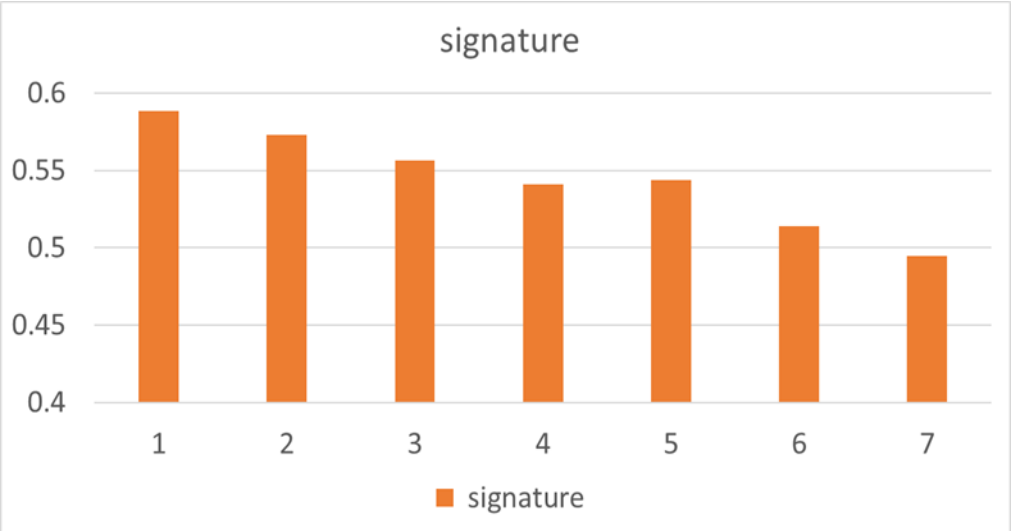

Figure 6.2: Neighbourhood size for signature similarity

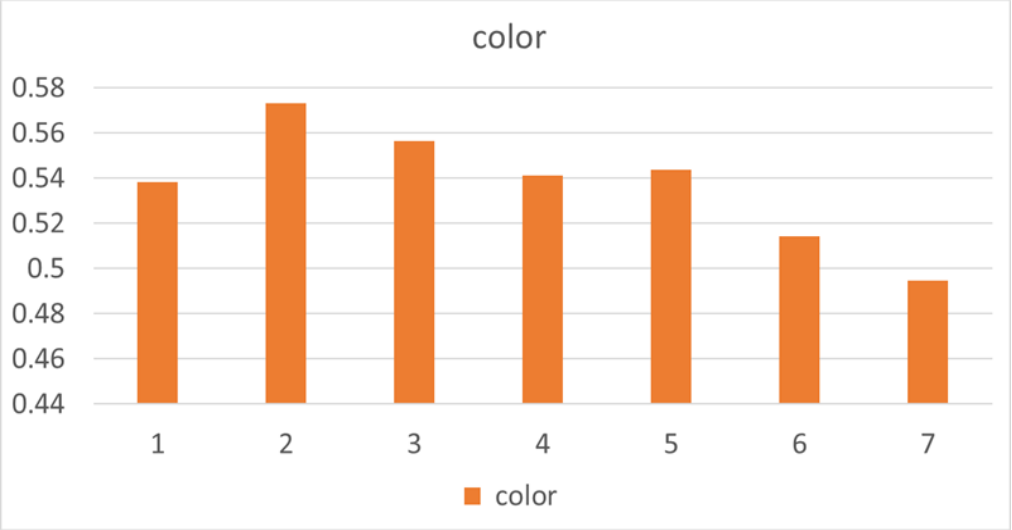

Figure 6.3: Neighbourhood size for color similarity

In all of the grids used for the following comparisons of iteration count and run time, the initial setting of neighbourhood size was set at 4. The neighbourhood size for the algorithm to use was gradually decreased every quarter of the total iterations count for the specific settings. These are the setting that performed the best in our tests under the time constraints given and in conjunction with other parameters setting.

Following are charts displaying the grid stress function over algorithm iteration counts from 10 000 up to 100 000.

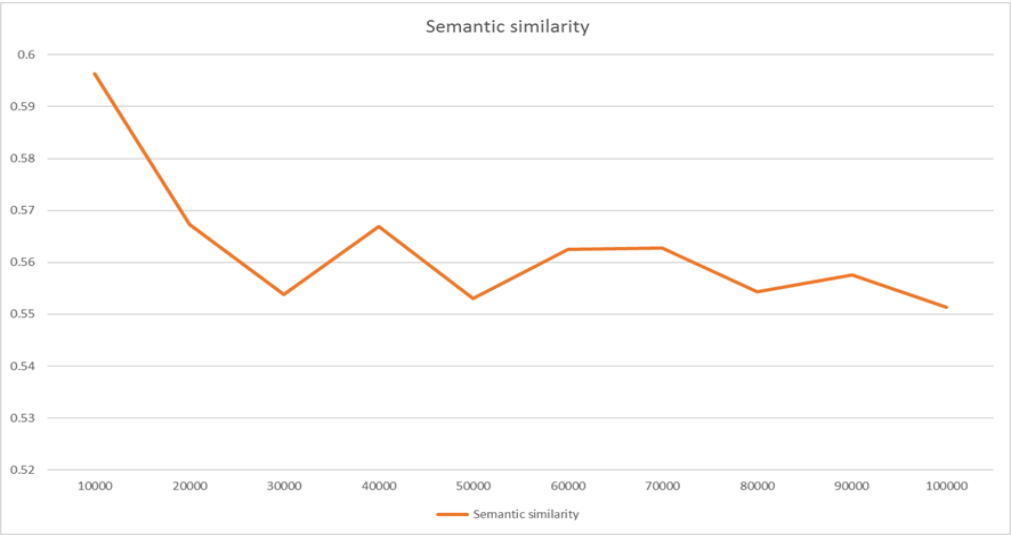

Figure 6.4: Semantic similarity stress average for different iteration counts

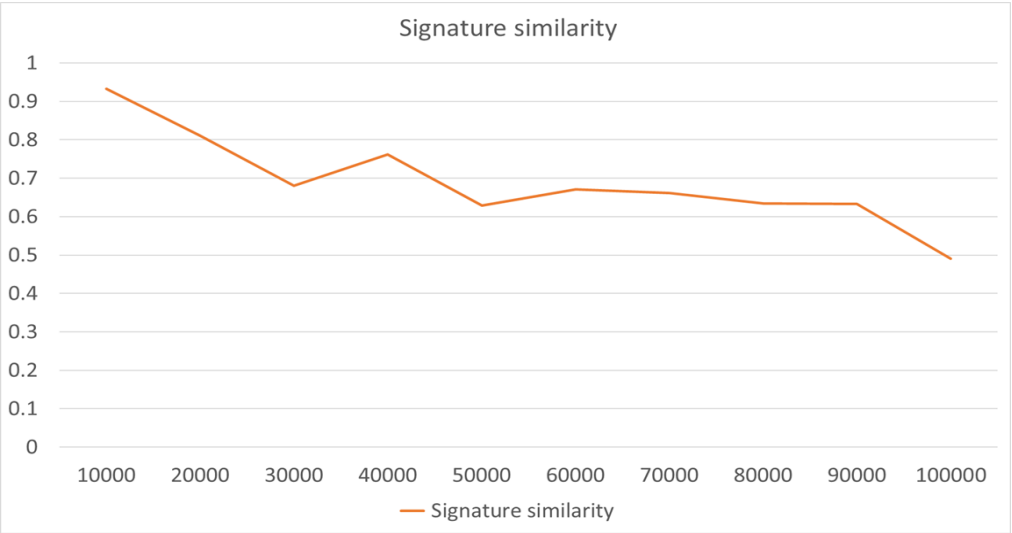

Figure 6.5: Signature similarity stress average for different iteration counts

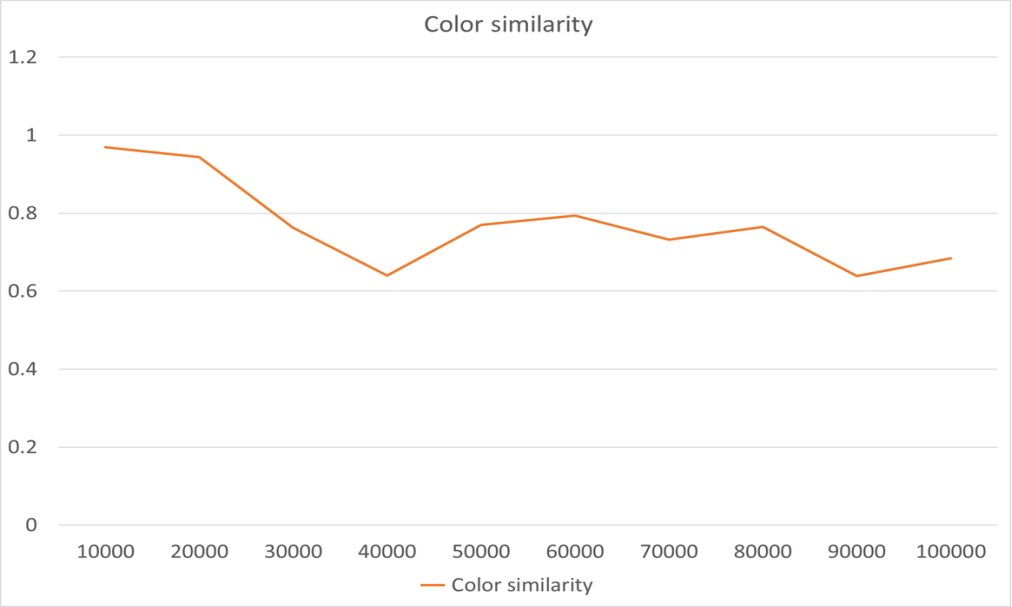

Figure 6.6: Color similarity stress average for different iteration counts

In figures 6.4, 6.5 and 6.6 is displayed the absolute value of stress function for the visualisations generated. Although it represents the overall quality of the grid in regards to the similarity sum, its problem is that different tested sets of key-frames begun at different value. That could influence the final value. So figures 6.7, 6.8, 6.9 display the evolution of the difference between initial stress value and final stress value of the algorithm for the same dataset.

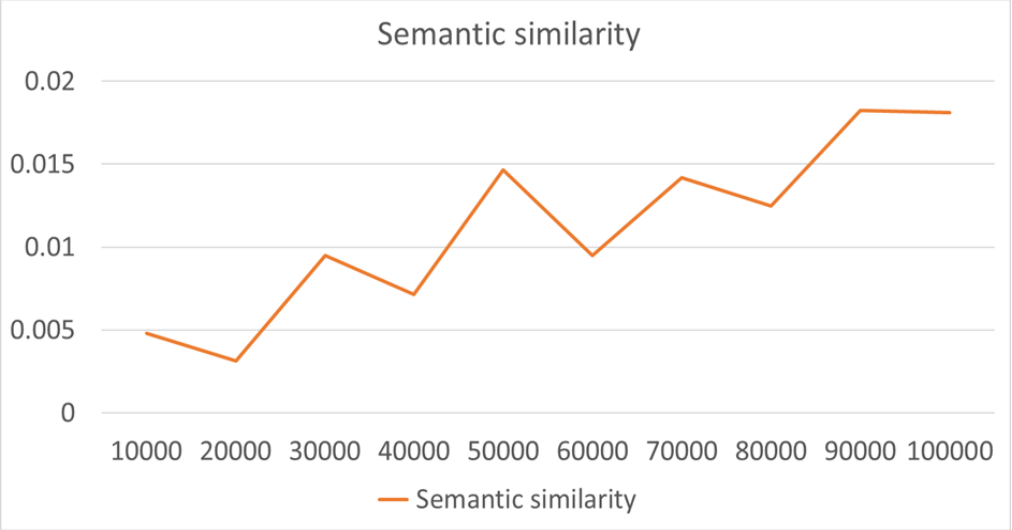

Figure 6.7: Semantic similarity stress average difference before and after organising the grid for different iteration counts

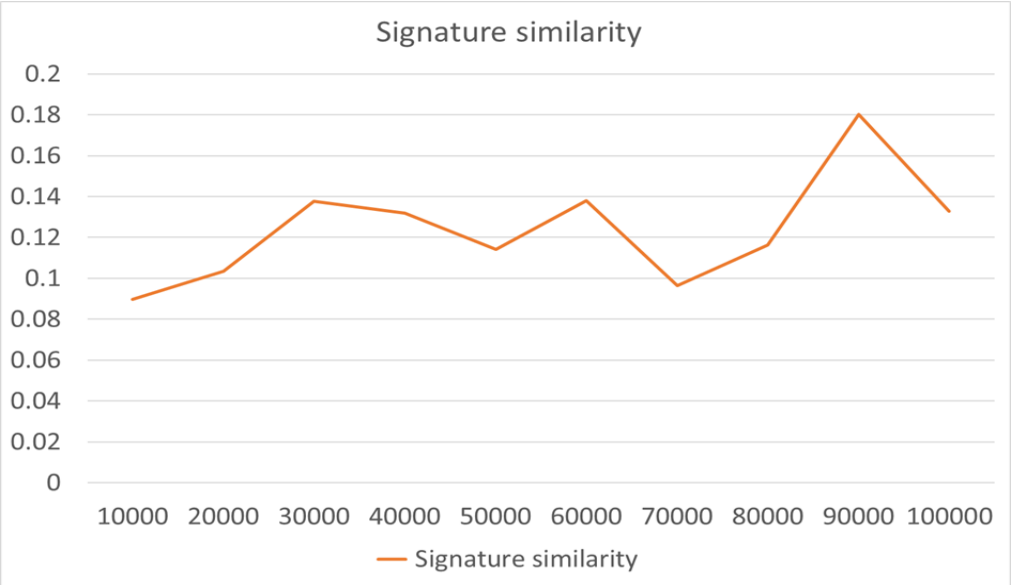

Figure 6.8: Signature similarity stress average difference before and after organising for different iteration counts

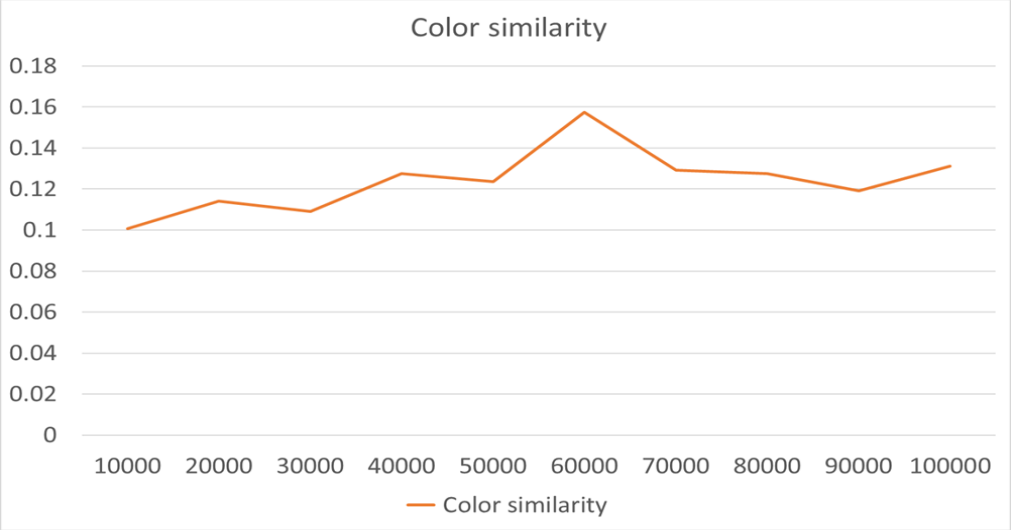

Figure 6.9: Color similarity stress average difference before and after organising for different iteration counts

We can clearly see decreasing tendency in the average final value of stress function over with increasing iterations count, respectively decreasing tendency int the stress value improvement during organisation by our algorithm. However, the improvements are not so significant and definitely not as consistent as we would expect them to be. There can be seen even quite significant deterioration in some specific iteration counts, especially in the semantic similarity metric.

6.2 Summary

The quality of the grid generated by proposed algorithm improves when increasing the number of iterations it runs, but not as significantly and consistently as we would like it to. The reason we see behind this is that the higher iteration number is not so necessary for smaller grid dimensions, because once these smaller grids get organised well enough, they can be only slightly improved and most of the added iterations are actually wasted.

From the data displayed in the charts, we deduced that the biggest difference in the final grid is between 10 000 and 40 000 iterations. On the other hand, with increasing neighbourhood size, the grid stress cost decreases significantly for every increase from size 1 to size 8.

That suggests, that the best approach to take in setting the parameters is to set the neighbourhood as big as possible and after fixing the neighbourhood size, setting the iteration count to the highest possible number to meet the time constraint.

To specify concrete value settings, on our testing machine, the highest configuration of parameters that is able to run in times under 500ms on average is neighbourhood size of 6 and 40 000 iterations. In the user study following, we will try to prove that this setting works well for majority of users.

7. User Study

To benchmark the quality of our methods and to determine the optimal values of various parameter involved, we performed a broad user study with our displays. We then compare the study to the experimental results gathered and try to find some form of correlation between them.

The whole study was conducted online as a web application.

It is currently located on the URL:
http://www.videobrowser.cz (as of 21.7.2017).
It will remain on that domain at least until September 2017. The source codes of the we application developed for the user study are attached to this thesis on the supplied data disc.

7.1 Methodology

300 image grids generated by the software tool were chosen for the user study. The grids were chosen in a way to reflect many possible settings of the algorithm parameters. For each of these grids, there was selected a key-frame located in that grid to query for in the user study. This key-frame was then presented to the user as a task, to find it in the grid it was taken from.

More details about the conditions and rules of the study can be on the web page of the study, when reading its "guidelines" (located on the second site in the work-flow, after clicking the "Bring it on!" button).

The grids were chosen such that:

- 100 grids use semantic similarity measure
- 100 use simple color RGB L2 distance measure
- 100 use color signature distance

In all 3 categories, the grids of multiple different sizes and aspect ratios were chosen, as well as with different algorithm settings. The setting that is emphasized to find the optimal value for is the iterations number. The second parameter we were changing to evaluate different setting was the initial neighbourhood size. For all the grids generated, gradual lowering of the neighbourhood size during the run time was used (as explained in previous chapter).

Every user was gradually chosen 10 random grids from the database. For each of them, the selected key-frame to look for was displayed for 3 seconds. Afterwards, the key-frame was hidden and the grid displayed.

The time from displaying (the time when the grid was actually rendered on-screen) until finding and clicking the image user selected was measured[1].

For evaluation of the results, only the search tasks in which the user selected the sufficiently similar image to the example one displayed were incorporated into the results. The formulation "sufficiently similar" instead of just "correct" is in place, because the grids usually contain more very similar key-frame of scenes from the same video, so we do not make any difference between 2 almost identical frames.

To choose the images that are sufficiently close to the target, all 3 metric tested (one distance for semantic, one for color and one for signatures) were calculated between the query image and the selected image. Results with over 66% similarity with the example were considered correct. That means, using normalised values between 0 and 1 for all the distance functions, only results with sum of similarities under 1 were considered correct.

The rate of the images chosen correctly for different grid generation methods were also compared.

7.2 Participants

The total of 263 people participated in the study through the web application providing us with 1264 queries answered. The total query count does not correspond to the number of participants multiplied by number of queries given to a participant because not everyone completed all 10 queries given. However, every single answer was saved into the database, so even the answers from unfinished runs can be utilized.

Out of all the queries, 172 were discarded because the time it tool to finish them was too high, so they could be considered outliers in context of our study. Every result that took more than 30s was discarded, but most of these took much longer time suggesting that the participant undertaking them probably left during the measurement, or simply did not take the study seriously.

7.3 Results

Checking our indicator of correctness for all of the meaningful answers, from the 1092 corrected answers, 387 were incorrect leaving us with 709 correct answers. The total success rate of 64.9% over the whole dataset does not give any really relevant information. However, the following table summarizing the answers for different grid generation methods could be much more interesting:

[1]We tried to measure the time as correctly as possible, starting the stopwatch measurement in JavaScript when the image of the grid was actually loaded. However, we have no guarantee when exactly is the "loaded" event fired and there could be still some delay between starting the stopwatch and actually displaying the grid. Nevertheless, its magnitude should not be too significant assuming averagely fast internet connection.

Metric	Total	Correct	Incorrect	Ratio of correct ones
Semantic similarity	385	259	126	67,27%
Color similarity	347	182	165	52,44%
Signatures similarity	360	268	92	74,4%

So the best ratio of correctly answered queries have signature similarities. Because the whole study was based only on visual observation based questions, it makes some sense. It has already been shown, that the signature similarities represent nicely images [4]. They were followed closely by semantic similarity measure and the worst rate of correct answers has shown the simple color similarity metric.

However, the differences were not so radical, and considering also the not so high count of respondents, it would be difficult to make some final conclusions based on this data. The more important data gathered by our study are the duration of individual searches. Only data from the correctly answered questions are included in the following tables:

Metric	Average answer time
Semantic similarity	14.7s
Color similarity	15.5s
Signatures similarity	14.2s

The result kind of surprisingly (compared to previous result) shows that the results for different metrics used are mostly comparable. Their differences are in the margin of error.

The next parameter to examine is the generated grid size. The following averages are calculated exclusive the higher bound and inclusive the lower bound:

Grid dimensions	Average answer time
Smaller than 8x8	8.7s
Between 8x8 and 12x12	14.2s
Between 12x12 and 16x16	15.8s
Between 16x16 and 20x20	22.7s
Bigger than 20x20	27.4s (many values over 30s)

As expected, the smaller the grid dimensions are (therefore the bigger the individual images are), the easier it is for a user to find and recognise the image he is looking for. Besides the size if the image, it is also caused by the fact, that the higher the dimensions, therefore the more images displayed, the more important it is or the grid to be well organised. And as int the study were used both well organised grids and grids sustaining of randomly selected images (even though oraganised by our algorithm, absolutely random ke-frames from a large video collection will always keep a certain level of disorganisation).

8. Implementation

We have implemented a video browsing tool based on the presented methods. Its primary purpose is their experimental evaluation and its working name is "VHVisualisation". After that, we have fine tuned the software to the best parameters we were able to set on our personal computer for the constrains given, that is grid calculation in under 500ms on average personal computer.

The application is developed in C# programming language using the Microsoft Visual Studio Enterprise 2017 IDE as the development environment. It is divided into 4 distinct modules:

- Client
- Search engine
- ImageMap (the visualisation)
- Extraction

Following appears the code map showing the primary classes and their dependencies created in the Visual Studio project in figure 8.1.

8.1 Client

The GUI of the application is built using the WPF framework. It utilized the combination of the C# language and XAML for defining the GUI client parts. Although the general idea behind WPF and XAML is to divide the rendering and logic part of the application, it is not so useful for the experimental purposes of our work. So the code responsible for rendering the user interface is distributed among both, XAML and C#.

Specifically The Client contains the *UIhelpers* class in which are implemented some useful methods for altering the WPF GUI, primarily the *Grids*.

The main class of the Client is the *Homepage* class, that contains all the other GUI elements. They consist of:

- ImageMap - the primary field for visualisation (Image grid itself)
- UI grid - a grid containing the user interface (controls and information)
- VideoSlider - A simple visualisation of unfolded video frames in a sequence (included to supper the primary visualisation when useful)
- VideoViewer - A class intended for displaying the playback of the video around the chosen key-frame (not used at the moment, because the video data extracted do not contain enough data for the video to be played.

- History grid - A field showing simple history of the search queries

The communication from the classes contained in *Homepage* backwards is implemented using the events system. The .NET native events for clicking the buttons and so on are redirected forward when necessary and custom events are created for the cases in which it is necessary.

The most essential interface of the Client are the methods:

- *GenerateRandomFrames* - used to create a random grid of key-frames from the loaded data
- *SeearchSimilar* - used to search the video database for the KNN of the query (passed as a parameter). The query can consist of more key-frames than one, in which the average of their feature vectors is used

8.2 Search Engine

The software tool implements a very simple search engine for video database querying. The single class *VideoDB* also manages storing the video data loaded in-memory. All other components that need access to the video data communicate with it.

Included in the Search Engine module are also the *Distances* used. They follow the strategy pattern, where all the specific distance implementations are classes implementing the abstract *DistanceCalcuklator* class. They all must have only one function in their interface: *GetDistance* that returns the distance between 2 *Frame* objects using supplied metric as a *double*.

8.3 Visualisation

The ImageMap module contains the primary part of the software, the visualisation system and algorithm. The primary class here is the *ImageMapGenerator* class that contains the methods for generating the grid, as well as to manipulate it and communicate with it from outside (such as export the grid or get information about the individual cells).

The function *GenImageMap* runs the whole algorithm for Image Map generation using the previously set settings and manages all the other parts of the algorithm. It accepts 2 parameters, the Metric (or distance function) to be used and the List of *Frame* objects to be sorted.

There are 2 other interesting classes, *ShotDetection* and *EnergyCalculator*. The *EnergyCalculator* is used by the Image Map generating algorithm itself to measure the overall grid stress. It calculated the grid stress function (or grid energy) on request by *CalculateEnergy* function that returns void. The energy as *double* can be then accesses through the *EnergyCalculator's* property *DisplayEnergy*.

Besides these, there is also the *ChartGenerator* class. It is used to generate charts representing the grid stress improvements over the algorithm run-time. It can also export the information about the grids generated into a text file.

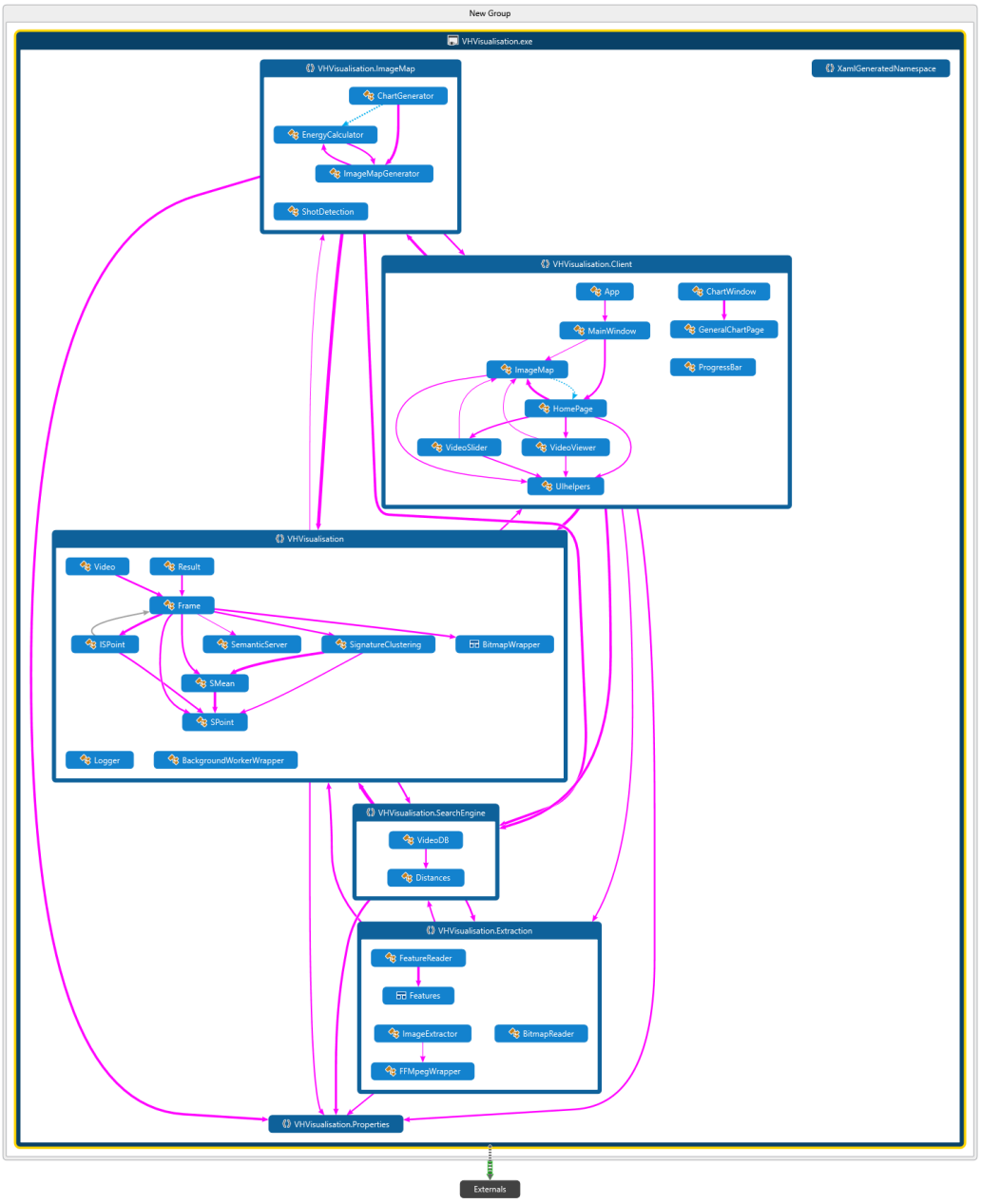

Figure 8.1: VHVisualisation code map

8.4 Extraction

For extracting bitmap images from the videos, FFmpeg.exe application is run in a separate process. For extracting the feature vectors using the neural network,

an external application "feature.exe" is run. It uses the VGG [5] network model compiled using the Caffe framework [6]. The whole feature extractor is used as a standalone external application.

There are also 2 classes with very specific purpose, the *BitmapReader* and the *FeatureReader*. Their purpose is to read and manage the data previously extracted for the TrecVid dataset used in all the experiments. I hereby thank my coworker Gregor Kovalczik for extraction the bitmap images and DCNN feature vectors from this dataset, as well as for creating these readers.

We are planning to develop the software tool attached further, as it is currently only in the development phase offering very little video search features. It is also not entirely production ready regarding the stability and reliability of the implementation because its primary purpose is experimental for further research of video search visualisation.

8.5 The user study web application

For the user study presented in chapter 7, a web application able to display the generated grids and select single images in it was developed. The application lets the users to answer a series of tasks in a game-like way, being shown an image and then tasked to find it in the grid displayed.

The application is written in object-oriented PHP with front-end utilizing HTML5 and JavaScript. The results are saved into a MySQL database. The application source codes with comments in the essential parts is attached to this work.

Conclusion

In the thesis, we have proposed an innovative method for visualising video search results for interactive browsing purposes. Current state-of-the art video browsing tools were discussed and we reasoned why there is need for improvement in the interactive search field. We analysed the problem of visualisation for interactive video browsers and based on our hypotheses, we designed the visualisation model and algorithm for its generation from search query results.

We have implemented the algorithm in a simple experimental video browser. The application offers basic functionality to search in video database and visualise video content in the image grid using the proposed methods. It is possible to interactively browse in the collection afterwards.

The algorithm was then experimentally evaluated by generating many grids and varying the parameters involved. The algorithm improves the stress function of the grids as expected. We chose the ideal setting of the parameters for the machines tested in order to find the compromise between visualisation quality and the time it takes to generate them.

The generated grids were also evaluated in a user study conducted. The results of the study generally support our hypotheses about the implemented algorithm. They also show what the preferred distance metrics and parameter settings of the users were. These correspond well with the exact results of the study - the measured search times of the tasks users were given. More details about the specific values can be found in the chapters about the study and experiments.

The work proved that good quality visualisation does make a significant difference in effectivity of browsing large video collections. We have shown that the presented algorithms do work and improve the search times in comparison to simpler visualisation methods, such as simple key-frames unfolding. We plan to compare our methods more directly with the competition be taking part in the video search competitions presented in chapter 3.

We see great potential in the interactive video exploration field and presume that we will be able to see more advance in its visualisation possibilities in future.

8.6 Future Work

There is a lot of space for improvement both, in interactive video browsing generally, and in the visualisation methods proposed in this work. As for the video browsing software tool, the retrieval model could be made more effective, implementing indexing methods and allowing to search the video collection more efficiently.

Regarding the visualisation itself, the first improvement we would like to make is to implement a Multilayer exploration structure (MLES) above our image grids. Currently, zooming is implemented only to allow interactively viewing the static image grid generated. We would like to allow fast interactive browsing through

the whole video collection. However, not through a statically generated MLES proposed in [13], but to generate this structure dynamically. When the user zooms into the grid, there would be individual QBE presented to the database for every image in the zoomed-in view, and its first result could be displayed instead if that single image that was used to create the query.

Bibliography

[1] Jochen Huber Klaus Schoeffmann, Marco A. Hudelist. Video interaction tools : A survey of recent work. *ACM Computing Surveys, Volume 48, Issue 1*, 2015.

[2] FFmpeg Developers. *FFMpeg [Software]*. 2016.

[3] T. Skopal J. Lokoč, A. Blažek. Signature-based video browser. *MultiMedia Modeling. Lecture Notes in Computer Science*, 8326, 2014.

[4] T. Skopal F. Matzner J. Lokoč, A. Blažek. Enhanced signature-based video browser. *MultiMedia Modeling. Lecture Notes in Computer Science*, 8936, 2015.

[5] A. Zisserman K. Simonyan. Very deep convolutional networks for large-scale image recognition. *arXiv*, 1409.1556, 2015.

[6] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv*, 1408.5093, 2014.

[7] Bo Gun Park, Kyoung Mu Lee, and Sang Uk Lee. A new similarity measure for random signatures: Perceptually modified hausdorff distance. 2006.

[8] Bernd Muenzer Stefan Petscharnig Christof Karisch1 Qing Xu Klaus Schoeffmann, Manfred Jurgen Primus and Wolfgang Huerst. Collaborative feature maps for interactive video search. *MultiMedia Modeling. Lecture Notes in Computer Science*, 2017.

[9] Algeron Ip Van Chin Manfred J. Primus Wolfgang Hurst, Klaus Schoeffmann. Storyboard-based video browsing using color and concept indices. *MultiMedia Modeling. Lecture Notes in Computer Science*, 2017.

[10] D. Kuboň A. Blažek, J. Lokoč. Video hunter at vbs 2017. *MultiMedia Modeling. Lecture Notes in Computer Science*, 2017.

[11] R. Mackowiak K. U. Barthel, N. Hezel. Navigating a graph of scenes for exploring large video collections. *MultiMedia Modeling. Lecture Notes in Computer Science*, 9517:418–423, 2016.

[12] Julie E. McCredden John D. Bain Graeme S. Halford, Rosemary Baker. How many variables can humans process? 2005.

[13] Mackowiak R Barthel K.U., Hezel N. Graph-based browsing for large video collections. *MultiMedia Modeling. Lecture Notes in Computer Science*, 8936, 2015.

[14] Nicholas Wyatt Kai Uwe Barthel, Steve Mammani. Automatic image sorting using mpeg-7 descriptors. *Proceedings of ICOB 2005*, 2005.

[15] Mackowiak R. Barthel K.U., Hezel N. ImageMap - Visually browsing millions of images. *MultiMedia Modeling. Lecture Notes in Computer Science*, 8936, 2015.

[16] G.E. Hinton L.J.P. van der Maaten. Visualizing high-dimensional data using t-sne. 2008.

[17] J.B. Kruskal and M. Wish. *Multidimensional Scaling.* Sage, 1978.

[18] Mehrotra R Ferman A M Warnick J Napbade, M. A high-performance shot boundary detection algorithm using multiple cues. *International Conference on Image Processing*, 1998.

[19] J. Lokoč D. Kuboň, A. Blažek. Multi-sketch semantic video browser. *MultiMedia Modeling. Lecture Notes in Computer Science, vol 9517*, 1917:406–411, 2016.

[20] Klaus Schoeffmann. A user-centric media retrieval competition: The video browser showdown 2012-2014. *IEEE MultiMedia*, 21, 2014.

[21] J. Lokoč M. Kruliš and T. Skopal. Efficient extraction of clustering-based feature signatures using gpu architectures. *Multimedia Tools and Applications*, pages 1–33, 2015.

[22] G. E. Hinton A. Krizhevsky, I. Sutskever. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems 25: Proceedings of 26th Annual Conference on Neural Information*, 25:1097–1105, 2012.

[23] Schulze S. Hughes J.Mn Blake, B.F. Perceptual mapping by multidimensional scaling: A step by step primer. 2003.

[24] Stephen D. Goldinger Michael C. Hout, Megan H. Papesh. Multidimensional scaling. *Wiley Interdiscip Rev Cogn Sci.*, pages 93–103, 2014.

List of Figures

List of Abbreviations

KIS - Known-item-search
QBE - Query by example
KNN - k nearest neighbours
CBIR - Content-base image retrieval
PMHD - Perceptually modified Hausdorff distance
DCNN - Deep Convolutional neural network
MLES - Multi-layer Exploration Structure

Glossary

Display - The term "display" is used for describing the rectangular image grids

Attachments

a DVD containing:

- The software tool created
- The source codes of the software
- The source codes of the web application used for the user study
- This thesis in pdf format
- Example video collection in binarized format (set of video meta files)
- User documentation for the software tool "VHVisualisation"